Loan Application Status Prediction

Problem Statement

In this every need money to survive. Money isn't everything but everything needs money. To earn money investment is must. Here comes the loan. People apply for in banks and loan distribution is core part of every banks business. Nobody can deny this fact that the main asset of bank is loan because from loan they earn a lot, however risk is major part of the loan. So, every banking company or farm always wants that their money should go in safe hands. That is why in today's world every company have their own process of verification of that person and validation, still there is not 100% guarantee that the customer will repay the loan amount or not. Here machine learning comes into picture, ML is much capable to identify that whether the loan can be given to applicant or not. In machine learning, machine learn each and every aspect of the data and then tells whether the loan can be given or not but in other hand in banks sometimes some situation comes when bank have to give loan to applicant of single strong factor, which is not possible because the frequency of these cases will be one in thousands or more applications.

Machine always helps the peoples it reduces the risk as well. Prediction of loan approval is very helpful for bank employees. The purpose of this blog is do provide good, fast and effective way to choose deserving candidate who applied for the loan. This model will save time and efforts of bank employee. Here first model will lean the data and then when test data will be provided to the model, it'll predict and tell whether loan should be given to applicant or not.

The dataset includes details of applicants who have applied for loan. The dataset includes details like credit history, loan amount, their income, dependents etc.

Independent Variables:

- Loan_ID

- Gender

- Married

- Dependents

- Education

- Self_Employed

- ApplicantIncome

- CoapplicantIncome

- Loan_Amount

- Loan_Amount_Term

- Credit History

- Property_Area

Dependent Variable (Target Variable):

-Loan_Status

The purpose of this blog is to build a model that can predict whether the loan of the applicant will be approved or not on the basis of the details provided in the dataset.

Data Analysis

Data Analysis is a procedure for gathering raw data than converting it into useful and informative data that will help for making decisions clear by the user. Data will be collect, analyzed to answer the questions.

```
In [2]: # Load and convert into dataframe
        data = pd.read_csv('loan_prediction[1].csv')
        # Looking first five rows
        data.head()
Out[2]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0

```
In [5]: # Checking shape of dataset
        data.shape
Out[5]: (614, 13)
In [6]: # we see that dataset have 614 rows and 13 columns
```

614 rows and 13 columns are available in this dataset to predict the Loan status.

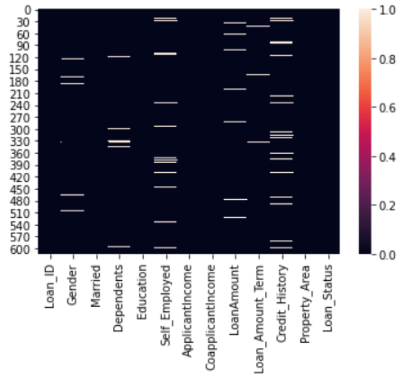
```
In [10]: #Checking the datatype of each variables
         data.dtypes

Out[10]: Loan_ID              object
         Gender               object
         Married              object
         Dependents           object
         Education            object
         Self_Employed        object
         ApplicantIncome       int64
         CoapplicantIncome   float64
         LoanAmount          float64
         Loan_Amount_Term    float64
         Credit_History      float64
         Property_Area        object
         Loan_Status          object
         dtype: object
```

Target variable i.e. Loan_Status is object so Classification will be used to learn model.

Exploratory Data Analysis

```
In [14]: #Looking for null values if any, in heatmap
         sns.heatmap(data.isna())
         plt.show()
```



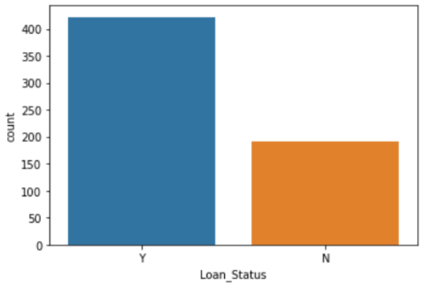
It's clearly visible that there are too many null values present in the dataset.

Removing Null values

```
In [21]: data['Gender']=data['Gender'].fillna(data['Gender'].mode()[0])
         data['Married']=data['Married'].fillna(data['Married'].mode()[0])
         data['Dependents']=data['Dependents'].fillna(data['Dependents'].mode()[0])
         data['Self_Employed']=data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
         data['LoanAmount']=data['LoanAmount'].fillna(data['LoanAmount'].mean())
         data['Loan_Amount_Term']=data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].median())
         data['Credit_History']=data['Credit_History'].fillna(data['Credit_History'].median())
```

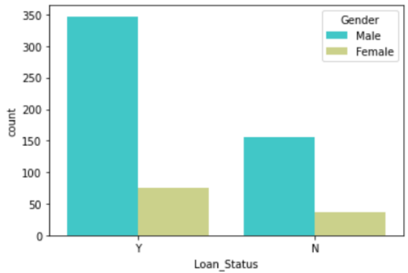
All the null values are removed from the datasets.

```
In [19]: sns.countplot(data['Loan_Status'])
         plt.show()
```



There are 422 Yes(loan approved) and 192 No (Not approved) values present.

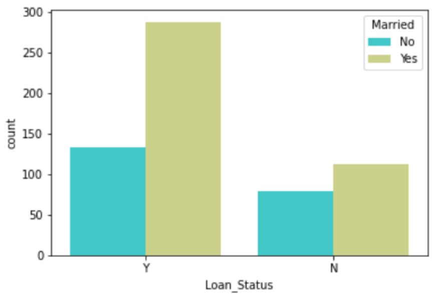
```
In [23]: sns.countplot(x='Loan_Status',hue='Gender',data=data,palette='rainbow')
         plt.show()
```




```
In [24]: #Loan of male applicants have more approved than female applicants
```

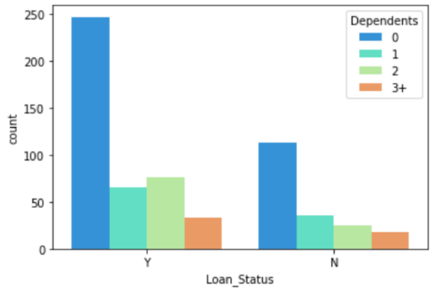
Here Males applicants are much more than Female

```
In [25]: sns.countplot(x=data['Loan_Status'],hue=data['Married'],palette='rainbow')
         plt.show()
```



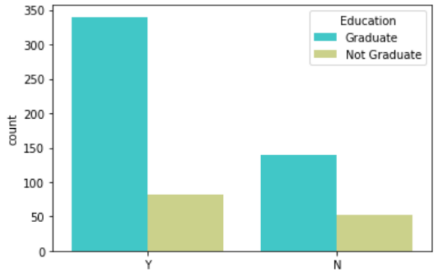
Loan of married applicants have more approved than unmarried applicants

```
In [27]: sns.countplot(x=data['Loan_Status'],hue=data['Dependents'],palette='rainbow')
         plt.show()
```



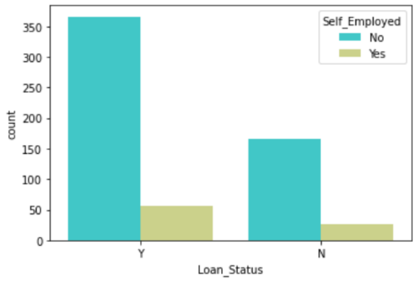
Loan of the applicants who have 0 dependents under them have approved the most.

```
In [29]: sns.countplot(x=data['Loan_Status'],hue=data['Education'],palette='rainbow')
         plt.show()
```



Loan of graduated applicants have more approved than notgraduated applicants

```
In [31]: sns.countplot(x=data['Loan_Status'],hue=data['Self_Employed'],palette='rainbow')
         plt.show()
```



Loan of self employed applicants have more rejected than approved in the dataset.

Label Encoding

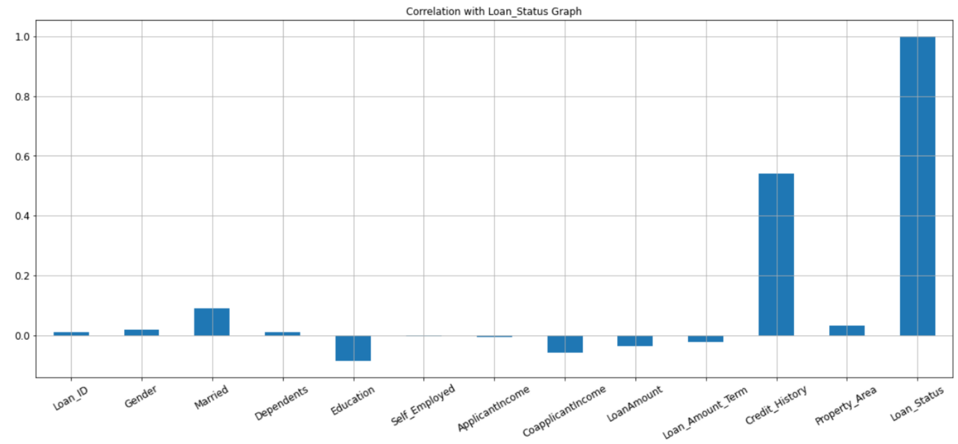
Let’s perform label encoding to convert object type columns into numeric type

```
In [41]: # now we seperate object datatype
         data_string_type=[]
         for i in data.columns:
             if data[i].dtypes == "object":
                 data_string_type.append(i)
```

```
In [42]: # Now we convert object into numerical
         from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
         for columns in data_string_type:
             data[columns]=le.fit_transform(data[columns])
```

Correlation with feature and label

```
In [49]: data.corrwith(data.Loan_Status).plot.bar(
                 figsize = (20, 8), title = "Correlation with Loan_Status Graph", fontsize = 12,rot = 30, grid = True)
         plt.show()
```

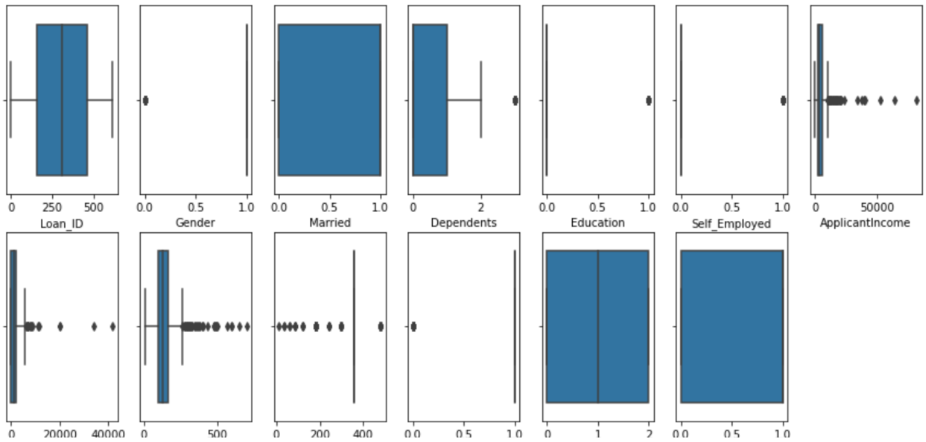


Loan Status is highly positively correlated to the credit history of applicants,and least relation of married and gender

CHECKING SKEWENESS

```
In [52]: # let's see how data is distributed for every column
         plt.figure(figsize=(15,7), facecolor='white')
         plotnumber = 1

         for column in data:
             if plotnumber<=13 :     # as there are 13 columns in the data
                 ax = plt.subplot(2,7,plotnumber)
                 sns.boxplot(data[column])
                 plt.xlabel(column,fontsize=10)
             plotnumber+=1
         plt.show()
```



we see that outlier present in dataset

Removing Skewness

```
In [55]: from scipy.stats import zscore
         z=np.abs(zscore(data))
         print(np.where(z>3))

(array([  9,  14,  68,  94, 126, 130, 133, 155, 155, 171, 171, 177, 177,
        183, 185, 242, 262, 278, 308, 313, 333, 333, 369, 402, 409, 417,
        432, 443, 487, 495, 497, 506, 523, 525, 546, 561, 575, 581, 585,
        600, 604], dtype=int64), array([7, 9, 9, 9, 6, 8, 9, 6, 8, 6, 8, 7, 8, 6, 6, 9, 9, 8, 8, 9, 6, 8,
        8, 7, 6, 7, 8, 6, 8, 9, 9, 8, 8, 8, 9, 8, 9, 7, 9, 7, 8],
       dtype=int64))
```

```
In [56]: threshold=3
         new_data=data[(z<3).all(axis=1)]
         print(data.shape)
         print(new_data.shape)

(614, 13)
(577, 13)
```

Building Machine Learning Models

Separating Input and Output Variables

```
In [64]: # Now we split feature and label
         x=data.drop("Loan_Status",axis=1)
         y=data["Loan_Status"]
```

DEAL WITH DATE IMBALANCE

```
In [65]: # Handling class imbalance using SMOTE
         from imblearn.over_sampling import SMOTE
         sm=SMOTE()
         x_over,y_over = sm.fit_resample(x,y)
```

Scaling

Scaling is required because there is too much difference in minimum and maximum value of columns.

```
In [66]: from sklearn.preprocessing import StandardScaler
         score =StandardScaler()
         X_score = score.fit_transform(x_over)

In [67]: #Checking for best random state which give best accuracy
```

Finding Best Random State

```
In [67]: #Checking for best random state which give best accuracy
         # To find the best random state using logistic Regressor model
         maxAccu=0
         maxRS=0
         for i in range(1,100):
             x_train,x_test,y_train,y_test=train_test_split(x_over,y_over,test_size=.30,random_state=i)
             mod= LogisticRegression()
             mod.fit(x_train,y_train)
             pred=mod.predict(x_test)
             acc=accuracy_score(y_test,pred)
             if acc>maxAccu:
                 maxAccu=acc
                 maxRS=i
         print ('best accuracy is',maxAccu,'on random state',maxRS)

         best accuracy is 0.7874015748031497 on random state 45

In [68]: x_train,x_test,y_train,y_test=train_test_split(x_over,y_over,test_size=.30,random_state=maxRS)
```

Train Test Split

Splitting train and test data, 70% data will be train and 30% data will be test

```
         best accuracy is 0.7874015748031497 on random state 45

In [68]: x_train,x_test,y_train,y_test=train_test_split(x_over,y_over,test_size=.30,random_state=maxRS)

In [69]: # Logistic model for training
```

Finding Best Algorithm

```
In [69]: # Logistic model for training
from sklearn.linear_model import LogisticRegression

log = LogisticRegression()

log.fit(x_train,y_train)

log_score =log.score(x_train,y_train)

print ('Logistic training Score ==>', log_score)

log_pred = log.predict(x_test)

log_cfm=confusion_matrix(y_test,log_pred)

log_accuracy = accuracy_score(y_test, log_pred)

print("Testing accuracy :", log_accuracy)

print(classification_report(y_test,log_pred))

log_cvs=cross_val_score(LogisticRegression(),x,y,cv=5).mean()
print("Cross_validation_score ----------",log_cvs)
log_Difference = (log_accuracy)*100 - (log_cvs)*100
print("Difference ----------",log_Difference)
```

```
Logistic training Score ==> 0.7186440677966102
Testing accuracy : 0.7874015748031497
              precision    recall  f1-score   support

           0       0.83      0.69      0.75       119
           1       0.76      0.87      0.81       135

    accuracy                           0.79       254
   macro avg       0.79      0.78      0.78       254
weighted avg       0.79      0.79      0.78       254

Cross_validation_score ---------- 0.8078368652538984
Difference ---------- -2.0435290450748766
```

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC

svc = SVC()

svc.fit(x_train,y_train)

svc_score =svc.score(x_train,y_train)

print ('SVC training Score ==>', svc_score)

svc_pred = svc.predict(x_test)

svc_cfm=confusion_matrix(y_test,svc_pred)


svc_accuracy = accuracy_score(y_test, svc_pred)


print("Testing accuracy :", svc_accuracy)


print(classification_report(y_test,svc_pred))

svc_cvs=cross_val_score(SVC(),x,y,cv=5).mean()
print("Cross_validation_score ----------",svc_cvs)
svc_Difference = (svc_accuracy)*100 - (svc_cvs)*100
print("Difference ----------",svc_Difference)
```

```
SVC training Score ==> 0.5508474576271186
Testing accuracy : 0.46062992125984253
              precision    recall  f1-score   support

           0       0.46      0.91      0.61       119
           1       0.45      0.07      0.12       135

    accuracy                           0.46       254
   macro avg       0.46      0.49      0.36       254
weighted avg       0.46      0.46      0.35       254

Cross_validation_score ---------- 0.6872984139677463
Difference ---------- -22.66684927079038
```

KNEIGHBORSCLASSIFIER

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(x_train,y_train)

knn_score = knn.score(x_train,y_train)

print ('KNeighborsClassifier training Score ==>', knn_score)

# Importing test set for prediction

knn_pred = knn.predict(x_test)

knn_cfm=confusion_matrix(y_test,knn_pred)

knn_accuracy =  accuracy_score(y_test,knn_pred)


print("Testing accuracy :", knn_accuracy)


print(classification_report(y_test,knn_pred))

knn_cvs = cross_val_score(KNeighborsClassifier(),x,y,cv=5).mean()

print("Cross_validation_score ----------",knn_cvs)
knn_Difference = (knn_accuracy)*100 - (knn_cvs)*100
print("Difference ----------",knn_Difference)
```

```
KNeighborsClassifier training Score ==> 0.7627118644067796
Testing accuracy : 0.562992125984252
              precision    recall  f1-score   support

           0       0.53      0.68      0.59       119
           1       0.62      0.46      0.53       135

    accuracy                           0.56       254
   macro avg       0.57      0.57      0.56       254
weighted avg       0.58      0.56      0.56       254

Cross_validation_score ---------- 0.6123950419832067
Difference ---------- -4.940291599895474
```

In [72]:

```python
# AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier

abc=AdaBoostClassifier()

abc.fit(x_train, y_train)

abc_score = (abc.score(x_train, y_train))

print('AdaBoostClassifier training Score ==>',abc_score)

# Importing test set for prediction

abc_pred = abc.predict(x_test)

abc_cfm=confusion_matrix(y_test,abc_pred)

abc_accuracy = accuracy_score(y_test,abc_pred)

print("Testing accuracy :", abc_accuracy)

print(classification_report(y_test,abc_pred))

abc_cvs = cross_val_score(AdaBoostClassifier(),x,y,cv=5).mean()

print("Cross_validation_score ----------",abc_cvs)
abc_Difference = (abc_accuracy)*100 - (abc_cvs)*100
print("Difference ----------",abc_Difference)
```

```
AdaBoostClassifier training Score ==> 0.8423728813559322
Testing accuracy : 0.7913385826771654
              precision    recall  f1-score   support

           0       0.81      0.73      0.77       119
           1       0.78      0.84      0.81       135

    accuracy                           0.79       254
   macro avg       0.79      0.79      0.79       254
weighted avg       0.79      0.79      0.79       254

Cross_validation_score ---------- 0.6906970545115287
Difference ---------- 10.064152816563663
```

In [73]:
```python
# DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier()

decision_tree.fit(x_train, y_train)

dt_score = (decision_tree.score(x_train, y_train))

print('Decision Tree training Score ==>',dt_score)

# Importing test set for prediction

dt_pred = decision_tree.predict(x_test)

dt_cfm=confusion_matrix(y_test,dt_pred)

dt_accuracy = accuracy_score(y_test,dt_pred)

print("Testing accuracy :", dt_accuracy)

print(classification_report(y_test,dt_pred))

dt_cvs = cross_val_score(DecisionTreeClassifier(),x,y,cv=5).mean()

print("Cross_validation_score ----------",dt_cvs)
dt_Difference = (dt_accuracy)*100 - (dt_cvs)*100
print("Difference ----------",dt_Difference)
```

```
Decision Tree training Score ==> 1.0
Testing accuracy : 0.7795275590551181
              precision    recall  f1-score   support

           0       0.73      0.85      0.78       119
           1       0.84      0.72      0.78       135

    accuracy                           0.78       254
   macro avg       0.79      0.78      0.78       254
weighted avg       0.79      0.78      0.78       254

Cross_validation_score ---------- 0.6954018392642942
Difference ---------- 8.412571979082387
```

In [74]:
```python
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()

rfc.fit(x_train, y_train)

rfc_score = (rfc.score(x_train, y_train))

print('RandomForest training Score ==>',rfc_score)

# Importing test set for prediction

rfc_pred = rfc.predict(x_test)

rfc_accuracy = accuracy_score(y_test,rfc_pred)

print("Testing accuracy :", rfc_accuracy)

rfc_cfm=confusion_matrix(y_test,dt_pred)

print(classification_report(y_test,rfc_pred))

rfc_cvs = cross_val_score(rfc,x,y,cv=5).mean()

print("Cross_validation_score ----------",rfc_cvs)

rfc_Difference = (rfc_accuracy)*100 - (rfc_cvs)*100
print("Difference ----------",rfc_Difference)
```

```
RandomForest training Score ==> 1.0
Testing accuracy : 0.9015748031496063
              precision    recall  f1-score   support

           0       0.94      0.84      0.89       119
           1       0.87      0.96      0.91       135

    accuracy                           0.90       254
   macro avg       0.91      0.90      0.90       254
weighted avg       0.91      0.90      0.90       254

Cross_validation_score ---------- 0.7801279488204719
Difference ---------- 12.144685432913434
```

```
In [76]: models = pd.DataFrame({'Classifier':['LogisticRegression', 'SVC', 'KNeighborsClassifier','AdaBoostClassifier','DecisionTreeClassi
                       'Score':[log_accuracy,svc_accuracy,knn_accuracy,abc_accuracy,dt_accuracy,rfc_accuracy,gnb_accuracy],
                        'CVS':[log_cvs,svc_cvs,knn_cvs,abc_cvs,dt_cvs,rfc_cvs,gnb_cvs],
                        'Difference':[log_Difference,svc_Difference,knn_Difference,abc_Difference,dt_Difference,rfc_Difference,gnb

         })
         models.sort_values(by='Score',ascending=False)
```

Out[76]:

	Classifier	Score	CVS	Difference
5	RandomForestClassifier	0.901575	0.780128	12.144685
3	AdaBoostClassifier	0.791339	0.690697	10.064153
0	LogisticRegression	0.787402	0.807837	-2.043529
4	DecisionTreeClassifier	0.779528	0.695402	8.412572
6	GaussianNB	0.779528	0.801333	-2.180524
2	KNeighborsClassifier	0.562992	0.612395	-4.940292
1	SVC	0.460630	0.687298	-22.666849

DecisionTreeClassifier have highest Accuracy more than 77.9 % and the difference between Cross Validation Score and Accuracy score it less. So DecisionTreeClassifier will be used here to learn model.

Hyper Parameter Tuning

Performing hyper parameter tuning to get good and more accurate result from the model

from sklearn.model_selection import **GridSearchCV**

```
In [88]: # Hyperparameter tunning the machine Learning Model

In [89]: grid_param = {"criterion": ["gini","entropy"],
                       "max_depth": range(2,10,3),
                       "min_samples_leaf": range(1,10,2),
                       "min_samples_split":range(2,10,2)}

In [90]: grid_Search = GridSearchCV(estimator = DecisionTreeClassifier(),param_grid = grid_param, cv = 5, n_jobs = -1)

In [91]: grid_Search.fit(x_train,y_train)

Out[91]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': range(2, 10, 3),
                                  'min_samples_leaf': range(1, 10, 2),
                                  'min_samples_split': range(2, 10, 2)})

In [92]: grid_Search.best_params_

Out[92]: {'criterion': 'gini',
          'max_depth': 8,
          'min_samples_leaf': 5,
          'min_samples_split': 2}
```

```
In [92]: grid_Search.best_params_
Out[92]: {'criterion': 'gini',
          'max_depth': 8,
          'min_samples_leaf': 5,
          'min_samples_split': 2}

In [93]: clf= DecisionTreeClassifier(criterion = 'gini',max_depth = 2, min_samples_leaf = 1, min_samples_split = 2)
         clf.fit(x_train,y_train)
Out[93]: DecisionTreeClassifier(max_depth=2)

In [94]: clf.score(x_train,y_train)
Out[94]: 0.747457627118644

In [95]: y_predict = clf.predict(x_test)

In [96]: accuracy_score(y_test,y_predict)
Out[96]: 0.7952755905511811

In [98]: #After hyperparatmeter tunning of of DecisionTreeClassifie is 79.5%

In [99]: #Saving the model
         import joblib
         joblib.dump(decision_tree,'loan_prediction')
Out[99]: ['loan_prediction']
```

CONCLUDING REMARKS

in above model it is summarised that relation analysis between features and target with best visualisation

also best model has been predicted with best hyperparameter tuning and best score was derived and made 79.5% accuracy